CONTROL METHOD FOR AN EXTENSIBLE MARKUP LANGUAGE FILE

Related Applications

This patent application is related to the United States patent application, Serial No. 09/419,217, entitled "Memory Management System and Method" filed on October 15, 1999, assigned to the same assignee as the present application and the United States patent application, Serial No. 09/768,102, entitled "Method of Storing a Structured Data Document" filed on January 23, 2001, assigned to the same assignee as the present application, the United States patent application, Serial No. 09/767,797, entitled "Method and System for Storing a Flattened Structured Data Document" filed on January 23, 2001, assigned to the same assignee as the present application and the United States patent application, Serial No. 09/768,101, entitled "Method of Performing A Search of a Numerical Document Object Model" filed on January 23, 2001, assigned to the same assignee as the present application and the United States patent application, Serial No. 09/767,493, entitled "Method of Operating an Extensible Markup Language Database" filed on January 23, 2001, assigned to the same assignee as the present application.

Field of the Invention

The present invention relates generally to the field of structured data documents and more particularly to an access control method for an extensible markup language file.

Background of the Invention

Structured data documents such as HTML (Hyper Text Markup Language), XML (eXtensible Markup Language) and SGML (Standard Generalized Markup Language) documents and derivatives use tags(metatags) to describe the data associated with the tags. This has an advantage over databases in that not all the fields are required to be predefined. XML is presently finding widespread interest for exchanging information between businesses. XML appears to provide an excellent solution for internet business to business applications. However, most companies need to limit access to data stored in XML documents. Unfortunately, present methods to limit access to data stored in an XML document are cumbersome.

Thus there exists a need for an access control method for an extensible markup language that is effective and easy to implement.

Brief Description of the Drawings

FIG. 1 is an example of an XML document in accordance with one embodiment of the invention;

FIG. 2 is an example of a flattened data document in accordance with one embodiment of the invention;

FIG. 3 is a block diagram of a system for storing a flattened data document in accordance with one embodiment of the invention;

FIG. 4 shows two examples of a map store cell in accordance with one embodiment of the invention;

FIG. 5 is a flow chart of a method of storing a structured data document in accordance with one embodiment of the invention;

FIG. 6 is a flow chart of a method of storing a structured data document in accordance with one embodiment of the invention;

FIG. 7 is a flow chart of a method of storing a structured data document in accordance with one embodiment of the invention;

FIG. 8 is a block diagram of a system for storing a flattened structured data document in accordance with one embodiment of the invention;

FIG. 9 is a block diagram of a system for storing a flattened structured data document in accordance with one embodiment of the invention;

FIG. 10 is a flow chart of the steps used in a method of storing a flattened structured data document in accordance with one embodiment of the invention;

FIG. 11 is a flow chart of the steps used in a method of storing a flattened structured data document in accordance with one embodiment of the invention;

FIG. 12 is a schematic diagram of a method of storing a numerical document object model in accordance with one embodiment of the invention;

FIG. 13 shows several examples of search queries of a numerical document object model in accordance with one embodiment of the invention;

FIG. 14 is a flow chart of the steps used in a method of performing a search of a numerical document object model in accordance with one embodiment of the invention;

FIG. 15 is a flow chart of the steps used in a method of performing a search of a numerical document object model in accordance with one embodiment of the invention;

FIG. 16 is a flow chart of the steps used in a method of translating a structured data document in accordance with one embodiment of the invention;

FIG. 17 is a flow chart of the steps used in a method of creating an alias in a numerical document object model in accordance with one embodiment of the invention;

FIG. 18 is a flow chart of the steps used in a method of operating an XML database in accordance with one embodiment of the invention;

FIG. 19 is a block diagram of a system for operating an XML database in accordance with one embodiment of the invention;

FIGs. 20A, B, and C are a flow chart of the steps used in a method of performing a search of an XML database in accordance with one embodiment of the invention;

FIG. 21 is an example of a convergence search query in accordance with one embodiment of the invention;

FIG. 22 is an example of an access control rule applied to a specific search query in accordance with one embodiment of the invention;

FIG. 23 is an example of an access control rule applied to a specific search query in accordance with one embodiment of the invention;

FIG. 24 is an example of an access control rule applied to a specific search query in accordance with one embodiment of the invention;

FIG. 25 is a flow chart of the steps used in an access control method in accordance with one embodiment of the invention;

FIG. 26 is a flow chart of the steps used in an control method in accordance with one embodiment of the invention;

FIGs. 27 & 28 are a flow chart of the steps used in an access control method in accordance with one embodiment of the invention;

FIG. 29 is an example of how a query is converted into an executable stack in accordance with one embodiment of the invention; and

FIG. 30 is a flow chart of the steps used in a control method in accordance with one embodiment of the invention.

Detailed Description of the Drawings

An control method for an extensible markup language file, includes the step of receiving a query from a user. Next, an access control rule associated with the user is determined. A query search on the extensible markup language file is performed. A query search result is stored. An access search on the extensible markup language file is performed. An access search result is stored. Finally, the query search result and the access search result are compared to determine an allowed search result. This system makes the access control as easy and as flexible as a search query.

FIG. 1 is an example of an XML document 10 in accordance with one embodiment of the invention. The words between the ‹› are tags that describe the data. This document is a catalog 12. Note that all tags are opened and later closed. For instance ‹catalog› 12 is closed at the end of the document ‹/catalog› 14. The first data item is "Empire Burlesque" 16. The tags ‹CD› 18 and ‹TITLE› 20 tell us that this is the title of the CD (Compact Disk). The next data entry is "Bob Dylan" 22, who is the artist. Other compact disks are described in the document.

FIG. 2 is an example of a flattened data document (numerical document object model) 40 in accordance with one embodiment of the invention. The first five lines 42 are used to store parameters about the document. The next line (couplet) 44 shows a line that has flattened all the tags relating to the first data entry 16 of the XML document 10. Note that the tag ‹ND› 46 is added before every line but is not required by the invention. The next tag is CATALOG› 47 which is the same as in the XML document 10. Then the tag CD› 48 is shown and finally the tag TITLE› 50.

Note this is the same order as the tags in the XML document 10. A plurality of formatting characters 52 are shown to the right of each line. The first column is the n-tag level 54. The n-tag defines the number of tags that closed in that line. Note that first line 44, which ends with the data entry "Empire Burlesque" 16, has a tag 24 (FIG. 1) that closes the tag TITLE. The next tag 26 opens the tag ARTIST. As a result the n-tag for line 44 is a one. Note that line 60 has an n-tag of two. This line corresponds to the data entry 1985 and both the YEAR and the CD tags are closed.

The next column 56 has a format character that defines whether the line is first (F) or another line follows it (N-next) or the line is the last (L). The next column contains a line type definition 58. Some of the line types are: time stamp (S); normal (E); identification (I); attribute (A); and processing (P). The next column 62 is a delete level and is enclosed in a parenthesis. When a delete command is received the data is not actually erased but is eliminated by entering a number in the parameters in a line to be erased. So for instance if a delete command is received for "Empire Burlesque" 16, a "1" would be entered into the parenthesis of line 44. If a delete command was received for "Empire Burlesque" 16 and ‹TITLE›, ‹/TITLE›, a "2" would be entered into the parenthesis. The next column is the parent line 64 of the current line. Thus the parent line for the line 66 is the first line containing the tag CATALOG. If you count the lines you will see that this is line five (5) or the preceding line. The last column of formatting characters is a p-level 68. The p-level 68 is the first new tag opened but not closed. Thus at line 44, which corresponds to the data entry "Empire Burlesque" 16, the first new tag opened is CATALOG. In addition the tag CATALOG is not closed. Thus the p-level is two (2).

FIG. 3 is a block diagram of a system 100 for storing a flattened data document in accordance with one embodiment of the invention. Once the structured data document is flattened as shown in FIG. 2, it can be stored. Each unique tag or unique set of tags for each line is stored to a tag and data store 102. The first entry in the tag and data store is ND›CATALOG›CD›TITLE› 104. Next the data entry "Empire Burlesque" 106 is stored in the tag and data store 102. The pointers to the tag and data entry in the tag and data store 102 are substituted into line 44. Updated line 44 is then stored in a first cell 108 of the map store 110. In one embodiment the tag store and the data store are separate. The tag and data store 102 acts as a dictionary, which reduces the required memory size to store the structured data document. Note that the formatting characters allow the structured data document to be completely reconstructed.

FIG. 4 shows two examples of a map store cell in accordance with one embodiment of the invention. The first example 120 works as described above. The cell (couplet) 120 has a first pointer (P_1) 122 that points to the tag in the tag and data store 102 and a second pointer (P_2) 124 that points to the data entry. The other information is the same as in a flattened line such as: p-level 126; n-tag 128; parent 130; delete level 132; line type 134; and line control information 136. The second cell type 140 is for an insert. When an insert command is received a cell has to be moved. The moved cell is replaced with the insert cell 140. The insert cell has an insert flag 142 and a jump pointer 144. The moved cell and the inserted cell are at the jump pointer.

FIG. 5 is a flow chart of a method of storing a structured data document. The process starts, step 150, by receiving the structured data document at step 152. A first data entry is determined at step 154. In one

embodiment, the first data entry is an empty data slot. At step 156 a first plurality of open tags and the first data entry is stored which ends the process at step 158. In one embodiment a level of a first opened tag is determined. The level of the first opened tag is stored. In another embodiment, a number of consecutive tags closed after the first data entry is determined. This number is then stored. A line number is stored.

In one embodiment, a next data entry is determined. A next plurality of open tags proceeding the next data entry is stored. These steps are repeated until a next data entry is not found. Note that the first data entry may be a null. A plurality of format characters associated with the next data entry are also stored. In one embodiment the flattened data document is expanded into the structured data document using the plurality of formatting characters.

FIG. 6 is a flow chart of a method of storing a structured data document. The process starts, step 170, by flattening the structured data document to a provide a plurality of tags, a data entry and a plurality of format characters in a single line at step 172. At step 174 the plurality of tags, the data entry and the plurality of format characters are stored which ends the process at step 176. In one embodiment, the plurality of tags are stored in a tag and data store. In addition, the plurality of format characters are stored in map store. The data entry is stored in the tag and data store. A first pointer in the map store points to the plurality of tags in the tag and data store. A second pointer is stored in the map store that points to the data store. In one embodiment, the structured data document is received. A first data entry is determined. A first plurality of open tags preceding the first data entry and the first data entry are placed in a first line. A next data entry is determined. A next plurality of open tags

proceeding the next data entry is placed in the next line. These steps are repeated until a next data entry is not found. In one embodiment a format character is placed in the first line. In one embodiment the format character is a number that indicates a level of a first tag that was opened. In one embodiment the format character is a number that indicates a number of tags that are consecutively closed after the first data entry. In one embodiment the format character is a number that indicates a line number of a parent of a lowest level tag. In one embodiment the format character is a number that indicates a level of a first tag that was opened but not closed. In one embodiment the format character is a character that indicates a line type. In one embodiment the format character indicates a line control information. In one embodiment the structured data document is an extensible markup language document. In one embodiment the next data entry is placed in the next line.

FIG. 7 is a flow chart of a method of storing a structured data document. The process starts, step 180, by flattening the structured data document to contain in a single line a tag, a data entry and a formatting character at step 182. The formatting character is stored in a map store at step 184. At step 186 the tag and the data entry are stored in a tag and data store which ends the process at step 188. In one embodiment a first pointer is stored in the map store that points to the tag in the tag and data store. A second pointer is stored in the map store that points to the data entry in the tag and data store. In one embodiment a cell is created in the map store for each of the plurality of lines in a flattened document. A request is received to delete one of the plurality of data entries. The cell associated with the one of the plurality of data entries is determined. A delete flag is set. Later a restore command is received. The delete flag is

unset. In one embodimcnt, a request to delete one of a plurality of data entries and a plurality of related tags is received. A delete flag is set equal to the number of the plurality of related tags plus one. In one embodiment, a request is received to insert a new entry. A previous cell containing a proceeding data entry is found. The new entry is stored at an end of the map store. A contents of the next cell is moved after the new entry. An insert flag and a pointer to the new entry is stored in the next cell. A second insert flag and second pointer is stored after the contents of the next cell.

Thus there has been described a method of flattening a structured data document to form a numerical document object model (DOM). The process of flattening the structured data document generally reduces the number of lines used to describe the document. The flattened document is then stored using a dictionary to reduce the memory required to store repeats of tags and data. In addition, the dictionary (tag and data store) allows each cell in the map store to be a fixed length. The result is a compressed document that requires less memory to store and less bandwidth to transmit.

FIG. 8 is a block diagram of a system 200 for storing a flattened structured data document (numerical DOM) in accordance with one embodiment of the invention. The system 200 has a map store 202, a dictionary store 204 and a dictionary index 206. Note that this structure is similar to the system of FIG. 3. The dictionary store 204 has essentially the same function as the map and tag store (FIG. 3) 102. The difference is that a dictionary index 206 has been added. The dictionary index 206 is an associative index. An associative index transforms the item to be stored, such as a tag, tags or data entry, into an address. Note that in one

embodiment the transform returns an address and a confirmer as explained in the United States patent application, Serial No. 09/419,217, entitled "Memory Management System and Method" filed on October 15, 1999, assigned to the same assignee as the present application and hereby incorporated by reference. The advantage of the dictionary index 206 is that when a tag or data entry is received for storage it can be easily determined if the tag or data entry is already stored in the dictionary store 204. If the tag or data entry is already in the dictionary store the offset in the dictionary can be immediately determined and returned for use as a pointer in the map store 202.

FIG. 9 is a block diagram of a system 220 for storing a flattened structured data document (numerical DOM) in accordance with one embodiment of the invention. A structured data document 222 is first processed by a flattener 224. The flattener 224 performs the functions described with respect to FIGs. 1 & 2 to form a numerical DOM. A parser 226 then determines the data entries and the associated tags. One of the data entries is transformed by the transform generator 228. This is used to determine if the data entry is in the associative index 230. When the data entry is not in the associative index 230, it is stored in the dictionary 232. A pointer to the data in the dictionary is stored at the appropriate address in the associative index 230. The pointer is also stored in a cell of the map store 234 as part of a flattened line.

FIG. 10 is a flow chart of the steps used in a method of storing a flattened structured data document (numerical DOM) in accordance with one embodiment of the invention. The process starts, step 240, by flattening the structured data document to form a flattened structured data document (numerical DOM) at step 242. Each line of the flattened structured data

document is parsed for a tag at step 244. Next it is determined if the tag is unique at step 246. When the tag is unique, step 248, the tag is stored in a dictionary store which ends the process at step 250. In one embodiment a tag dictionary offset is stored in the map store. A plurality of format characters are stored in the map store. When a tag is not unique, a tag dictionary offset is determined. The tag dictionary offset is stored in the map store.

In one embodiment, the tag is transformed to form a tag transform. An associative lookup is performed in a dictionary index using the tag transform. A map index is created that has a map pointer that points to a location in the map store of the tag. The map pointer is stored at an address of the map index that is associated with the tag transform.

FIG. 11 is a flow chart of the steps used in a method of storing a flattened structured data document (numerical DOM) in accordance with one embodiment of the invention. The process starts, step 260, by receiving the flattened structured data document (numerical DOM) that has a plurality of lines (couplets) at step 262. Each of the plurality of lines contains a tag, a data entry and a format character. The tag is stored in a dictionary store at step 264. The data entry is stored in the dictionary store at step 266. At step 268 the format character, a tag dictionary offset and a data dictionary offset are stored in a map store which ends the process at step 270. In one embodiment, the tag is transformed to form a tag transform. The tag dictionary offset is stored in a dictionary index at an address pointed to by the tag transform. In one embodiment, it is determined if the tag is unique. When the tag is unique, the tag is stored in the dictionary store otherwise the tag is not stored (again) in the dictionary store. To determine if the tag

is unique, it is determined if a tag pointer is stored in the dictionary index at an address pointed to by the tag transform.

In one embodiment, the data entry is transformed to form a data transform. The data dictionary offset is stored in the dictionary index at an address pointed to by the data transform. In one embodiment each of the flattened lines has a plurality of tags.

In one embodiment, a map index is created. Next it is determined if the tag is unique. When the tag is unique, a pointer to a map location of the tag is stored in the map index. When the tag is not unique, it is determined if a duplicates flag is set. When the duplicates flag is set, a duplicates count is incremented. When the duplicates flag is not set, the duplicates flag is set. The duplicates count is set to two. In one embodiment a transform of the tag with an instance count is calculated to form a first instance tag transform and a second instance tag transform. A first map pointer is stored in the map index at an address associated with the first instance transform. A second map pointer is stored in the map index at an address associated with the second instance transform.

In one embodiment a transform of the tag with an instances count equal to the duplicates count is calculated to form a next instance tag transform. A next map pointer is stored in the map index at an address associated with the next instance transform.

In one embodiment, a map index is created. Next it is determined if the data entry is unique. When the data entry is unique, a pointer to a map location of the tag is stored.

Thus there has been described an efficient manner of storing a structured data document that requires significantly less memory than

conventional techniques. The associative indexes significantly reduces the overhead required by the dictionary.

FIG. 12 is a schematic diagram of a method of storing a numerical document object model in accordance with one embodiment of the invention. This is similar to the models described with respect to FIGs. 3 & 8. The couplets (flattened lines) are stored in the map store 302. A tag dictionary 304 stores a copy of each unique tag string. For instance, the tag string CATALOG›CD›TITLE› 306 from line 44 (see FIG. 2) is stored in the tag dictionary 304. Note that the tag ND› is associated with every line and therefor has been ignored for this discussion. A tag dictionary index 308 is created. Every tag, incomplete tag string and complete tag string is indexed. As a result the tag CATALOG› 310, CATALOG›CD› 312 and every other permutation is stored in the tag index 308. Since a tag may occur in multiple entries it may have a number of pointers associated with the tag in the index.

A data dictionary 314 stores a copy of each unique data entry such as "Bob Dylan". A data dictionary index 316 associates each data entry with its location in the dictionary. In one embodiment, the tag dictionary index and the data dictionary index are associative memories. Thus a mathematical transformation of the entry such as "Bob Dylan" provides the address in the index where a pointer to the entry is stored. In addition to the tag and data indices a map index 318 is created. The map index 318 contains an entry for every complete tag string (see string 306) and the complete tag string and associated data entry. Note that the map index may be an associative index. By creating these indices and dictionaries it is possible to quickly and efficiently search a structured data document. In addition, once the document is in this

form it is possible to search for a data entry without ever having to look at the original document.

FIG. 13 shows several examples of search queries of a numerical document object model in accordance with one embodiment of the invention. The first example 330 is a fully qualified query since a complete tag string has been specified. The second example 332 is also a fully qualified query since a complete tag string and a complete data entry have been specified. The third example is a not fully qualified query since a partially complete tag string has been specified. The fourth 336 and fifth 338 examples are also examples of a not fully qualified query since the data entry is not complete. Note that the * stands for any wild card. If the data entry were completely specified, the query would be fully qualified.

FIG. 14 is a flow chart of the steps used in a method of performing a search of a numerical document object model in accordance with one embodiment of the invention. The process starts, step 350, by receiving a query at step 352. When the query is a fully qualified query, the target is transformed to form a fully qualified hashing code at step 354. Note the phrase "fully qualified hashing code" means the hashing code for the target of a fully qualified query. In one embodiment the hashing code is a mathematical transformation of the target to produce an address and a confirmer as explained in the United States patent application, Serial No. 09/419,217, entitled "Memory Management System and Method" filed on October 15, 1999, assigned to the same assignee as the present application and hereby incorporated by reference. An associative lookup in a map index is performed using the fully qualified at step 356. At step 358, a map offset is returned. At

step 360, a data couplet is returned which ends the process at step 362. In one embodiment, an identified couplet of the numerical DOM (as stored in the map) is converted into an XML string. When the query is partially qualified, the target is transformed to form a partially qualified . An associative lookup is performed in a dictionary index using the partially qualified . A partially qualified query is one that does not contain a complete tag or data string, i.e, <TITLE> instead of ND>CATALOG>CD>TITLE>. A dictionary offset is returned. The complete string is located in the dictionary, using the dictionary offset. A pointer is located in a map index using the complete string. The complete reference is located in the numerical DOM using the pointer. The data couplet is converted into a data XML string.

In another embodiment, when the query includes a wildcard target, the dictionary is scanned for the wildcard target. A complete string is returned from the dictionary that contains the wildcard target. A pointer is located in a map index using the complete string. A couplet is located in the numerical DOM using the pointer.

In one embodiment the hashing code is determined using linear feedback shift register operation, such as (but not limited to) a cyclical redundancy code. In another embodiment, the hashing code is determined by using a modulo two polynomial division. In one embodiment, the divisor polynomial is an irreducible polynomial. Other hashing codes may also be used.

FIG. 15 is a flow chart of the steps used in a method of performing a search of a numerical document object model in accordance with one embodiment of the invention. The process starts, step 370, by receiving a query at step 372. A target type of the query is determined at step 374.

When the target type is an incomplete data string, a sliding window search of a dictionary is performed at step 376. An incomplete data string could be <Bob> instead of <Bob Dylan>. A dictionary offset of a match is returned at step 378. In one embodiment a plurality of dictionary offsets are returned. At step 380 an incomplete data couplet is returned which ends the process at step 382. When the target type is an incomplete tag and a complete data string, the incomplete tag is transformed to form an incomplete target . An associative lookup in a map index is performed using the incomplete tag . At least one map offset is returned. The complete data string is transformed to form a complete data string . An associative lookup is performed in the map index. A data string map offset is returned. Next, the at least one map offset is compared with the data string map offset.

FIG. 16 is a flow chart of the steps used in a method of translating a structured data document in accordance with one embodiment of the invention. The process starts, step 390, by creating a numerical DOM of the structured data document at step 392. A first format dictionary is translated into a second format dictionary at step 394. At step 396 a second set of dictionary pointers are added to the dictionary index. The second set of dictionary pointers point to the offsets in the second format dictionary which ends the process at step 398. In one embodiment, a plurality of dictionary offset pointers are converted to a plurality of dictionary index pointers. This converts the map so it points to the dictionary index rather than the offsets into the dictionary, since there are two dictionaries now.

FIG. 17 is a flow chart of the steps used in a method of creating an alias in a numerical document object model in accordance with one embodiment of the invention. The process starts, step 410, by receiving an alias request at step 412. A dictionary offset for the original string in a

dictionary is found at step 414. At step 416 the original string is converted to the alias at the dictionary offset which ends the process at step 418. An alias index is created that associates the alias and the original string or the dictionary offset of the original string, and in one embodiment the creation of the alias index includes creating an array that matches the dictionary offset to the original string. In another embodiment, the original string is transformed to form a string . An associative lookup in the dictionary is performed to find the dictionary offset.

A method of performing a search of a numerical document object model begins when the system receives a query. The query is transformed to form a fully qualified . An associative lookup is performed in a map index using the fully qualified . Finally, a map offset is returned. In one embodiment, an identified couplet of the numerical DOM is converted into an XML string. In another embodiment, it is determined if the target is a complete data string. When the target is a complete data string, the complete data string is transformed to form a complete . An associative lookup is performed in a dictionary index using the complete data . A dictionary offset is returned. The numerical DOM is scanned for the dictionary offset, and a data couplet is returned. In another embodiment the data couplet is converted into a data XML string. In another embodiment, the system determines if the target is a wildcard data string. When the target is the wildcard data string, performing a sliding window search of a dictionary. The system returns a dictionary offset of a match and scans the numerical DOM for the dictionary offset. An incomplete data couplet is returned.

FIG. 18 is a flow chart of the steps used in a method of operating an XML database in accordance with one embodiment of the invention.

The process starts, step 420, by receiving a structured data document at step 422. The structured data document is flattened to form a flattened document at step 424. At step 426 a data transform is created for each of a plurality of data entries. A tag string transform is created for each of a plurality of associated tags at step 428. At step 430 a pointer is stored in each of a plurality of cells of a map store which ends the process at step 432.

In one embodiment, a plurality of data entries and a plurality of tag entries are determined when the document is flattened. In another embodiment, the system stores a copy of each unique data entry in a data dictionary and then correlates the data transform to a data dictionary pointer in an associative data dictionary index. In another embodiment, first and second data dictionaries are created. The first and second data dictionaries are used to store first and second language copies of each unique data entry, respectively. The languages may be a computer-oriented format, such as ASCII or rich text, or the languages may be human, such as English or French. The data transform is correlated to a pair of dictionary pointers in the associative data dictionary index. A copy of each unique tag string is stored in a tag dictionary and the tag string transform is correlated to a tag dictionary pointer in an associative tag dictionary index. In another embodiment, first and second tag dictionaries are created. The first and second tag dictionaries are used to store first and second language copies of each unique tag entry, respectively. The tag transform is correlated to a pair of dictionary pointers in the associative tag dictionary index. Next an original entry and an alias entry are cross-referenced in an alias index.

In another embodiment, the system receives a search query. It is determined whether the search query contains a fully qualified target. When the search query does contain the fully qualified target, the fully qualified target is transformed to form a fully qualified transform. Next, a target pointer is received from the associative map index using the fully qualified transform, and the data couplet pointed to by the target pointer is read.

In another embodiment, the search query does not contain the fully qualified target. The partially qualified target is transformed to form a partially qualified transform. The system performs an associative lookup in the associative tag dictionary index using the partially qualified transform. The system returns a tag dictionary offset for the partially qualified transform, and a complete tag string is located in the tag dictionary. Next, the system receives a target pointer for the partially qualified transform, and the system reads the data couplet pointed to by the target pointer.

In another embodiment, the system receives an alias command containing an original element and an alias element, and an alias pointer is stored in an address of the alias index that is associated with the original entry. The alias element is transformed to form an alias transform and it is determined if the alias pointer is associated with the alias transform in the data dictionary index or the associative tag dictionary index. When the alias pointer is not associated with the alias transform, the alias element is stored in either the data dictionary or the tag dictionary and the alias pointer is returned. When the alias pointer is associated with the alias transform, the alias pointer is returned.

In another embodiment, the system receives a print command requesting a portion of the structured data document be printed in the second language. The system retrieves a first couplet from the portion of the map store and expands the first couplet using the second language data dictionary and the second language tag dictionary.

FIG. 19 is a block diagram of a system 440 for operating an XML and derivatives database in accordance with one embodiment of the invention. The system 440 receives a structured data document 442 at the document flattener 444. The document flattener 444 sends the flattened document to the transform generator 446, which creates a data transform for each of a plurality of data entries and a tag string transform for a plurality of associated tags. A map store 448 is connected to the transform generator and has a plurality of cells, each containing the data transform, the tag string transform and a format character. An associative map index 450 has a plurality of map addresses, each of the plurality of addresses having a pointer to the map store 448.

In one embodiment, the parser 452 receives the flattened document from the document flattener 444 and determines the plurality of data entries and the plurality of associated tags. In another embodiment, a data dictionary stores a copy of each unique data entry, and an associative data dictionary index 454 has a plurality of data addresses that correlates the data transform to a dictionary pointer.

In another embodiment, the data dictionary includes a first data dictionary 456 and a second data dictionary 458. The second data dictionary 458 stores the copy of each unique data entry in a second format. A data translation index 460 points to the first data dictionary 456 or the second data dictionary 458.

In another embodiment, a tag dictionary stores a copy of each unique tag string, and an associative tag dictionary index 462 has a plurality of tag addresses that correlates the tag string transform to a tag dictionary pointer. The tag dictionary includes a first tag dictionary 464 and a second tag dictionary 466, and the second tag dictionary 466 stores the copy of each unique tag string in a second format. A tag translation index 468 points to the first tag dictionary 464 or the second tag dictionary 466.

In another embodiment, an alias index 470 cross-references an original entry and an alias entry, and a search engine 472 is connected to the map store 448.

FIGs. 20A, B, and C are a flow chart of the steps used in a method of performing a search of an XML database in accordance with one embodiment of the invention. The process starts, step 480, when the system receives a query containing a first data target, a second data target and a convergence point at step 482. At step 484 the system determines a convergence level of the convergence point. The system performs a transform of the first data target and the second data target to form a first transform and a second transform at step 486, and at step 488 reads a first couplet containing the first data target using the map index. At step 490 the system reads a second couplet containing the second data target using the map index, and at step 492 it determines if a first p-level of a first couplet is greater than the convergence level, and when the first p-level is not greater than the convergence level, the system determines a line number for the first couplet at step 494. At step 496, when a second p-level of a second couplet is greater than the convergence level, the system determines if a parent p-level is greater

than the convergence level, and when the parent p-level is not greater than the convergence level, the system determines a line number of a parent line at step 498. At step 500, when the line number of the parent is equal to the line number of the first couplet, the system determines if a match is found, which ends the process at step 502.

In one embodiment, when the line number of the parent is not equal to the line number of the first couplet, the system determines that the match is not found. In another embodiment, when the first p-level is greater than the convergence level, scanning the successive parents to find a parent line with a parent p-level not greater than the convergence level. Next, the system determines is the line number of the parent line of the second couplet is equal to a line number of the parent line of the first couplet, and when the line numbers are equal, the system determines that a match had been found.

FIG. 21 is an example of a search query 510 in accordance with one embodiment of the invention. The search query 510 is searching for "Greatest Hits" 512 and "Dolly Parton" 514 converging at the tag <cd>. The first data entry "Greatest Hits" 512 has a <Title> tag entry 516. The second data entry "Dolly Parton" 514 is partially qualified because it has no tag entry. Referring back to FIG. 2, <cd> is a level 3 tag, and the first and second data entries are found in lines 17 and 18 respectively. Starting with the "Greatest Hits" search parameter on line 17, if the p-level of the line where the search term is located is not greater than the convergence level, the system ceases searching. For line 17, the p-level is 3 and the convergence level is 3, so line converges on itself. Next, the system searches for the second search query term, "Dolly Parton." "Dolly Parton" is found at line 18. The system compares the p-level of line 18, in this instance 4, to the

convergence level of the query, in this instance 3. The p-level of line 18 is 4, which is greater than the convergence level, 3. The system moves up to line 18's parent and determines the parent line's p-level. The parent line of line 18 is line 17, in this case. The p-level of the parent line, line 17 is 3, is not greater than the convergence level, 3. Next, the system compares the parent line's line number, 17, to the line number of the first query term, 17. Convergence occurs when these two line numbers are the same. Thus the convergence of "Greatest Hits" and "Dolly Parton" occurs under the tag <cd> at line 17.

Thus there has been described a method of operating an extensible markup language database that is significantly more efficient.

FIG. 22 is an example of an access control rule applied to a specific search query in accordance with one embodiment of the invention. The figure shows a query 600 and an access control rule 602. Note that it is assumed that the query is against the document of FIG. 1. The search query 600 asks that all "CDs" in the document be returned. From FIG. 2 we can see that new CDs start on lines 6, 12, 18, & 24 (604). However, a search query always returns a complete fragment. As a result, the query will return all the lines shown in block 606. The access control rule 602 is just a query against the document. The access control rule is used to block access to CDs with a price less than $10.00. Note that the systems is flexible enough to allow access rules with a predicate [price<$10.00] 608. This search will return lines 16 & 22 (see FIG. 2). However, the query always returns a complete fragment. As a result, these lines are expanded to the block of lines 610. An intersection between the block of lines 606 and the block of lines 610 defines the fragments for which a user is not allowed access. Note that the system is flexible enough that the search query could define the

fragments for which a user is allowed access. Note that the search querie's convergence depth (level) is 2 (612) while the convergence depth for the access control rule is 3 (614). For a more complex document the access control rule would have to be converged to the same depth as the query. In another embodiment, the p-level from lines returned from the access control rule can be compared to the p-level of the lines from the query. The system ensures that the searches are converged to the same p-level. Note that the phrase "performing an intersection between the search query result and the access search result" includes both methods described above.

FIG. 23 is an example of an access control rule applied to a specific search query in accordance with one embodiment of the invention. The example has a query 620 and an access control rule 622. This example differs from the one in FIG. 22 in that the convergence depths 624 & 626 are the same. In this case the search query is interested in the artist 628 of the CD. The access control rule is the same and returns lines 16 & 22. However, these lines are not complete fragments so they are expanded to the block of lines 628. This block of lines is intersected with the lines returned by the search query to determine the allowable data. Lines 13 & 19 intersect and therefore would not be returned to the user (assuming the access control rule is to exclude these results).

FIG. 24 is an example of an access control rule applied to a specific search query in accordance with one embodiment of the invention. Like the previous examples a query 640 and an access control rule 642 are shown. In this case the access control rule 642 has a convergence depth 644 that is lower than the convergence depth 646 of the query 640. In this case the access control results 648 are expanded to the same depth as the query results 650 to form the block of lines 652. Note that there is now a

complete intersection of query search results 650. In another embodiment, the search results are converged until they are at the same p-level as the access control results. Since line 7 then converges to line 6 and lines 13, 18, 24 converge to lines 12, 18, 24 respectively, the allowable search result is a null set. This assumes that the access rule is a hide.

FIG. 25 is a flow chart of the steps used in an access control method in accordance with one embodiment of the invention. The process starts, step 660, by performing an access search on the extensible markup language file to form an access search result at step 662. In one embodiment the access search is a query against the file and the result is lines of the file. A query search is performed on the extensible markup language file to form a query search result at step 664. In one embodiment, the query search result is a set of line numbers from a flatten extensible markup language file. The query search result may include a format character for each of the line numbers. At step 666 the access search result is compared to the query search result to form an allowed search result which ends the process at step 668. In one embodiment a search convergence depth is compared to an access convergence depth (level). When the search convergence depth is equal to the access convergence depth, an intersection between the query search result and the access search result is performed to form an intersection result. When the search convergence depth is greater than the access convergence depth, the query search result is converged to the access convergence depth to form a converged query search result. An intersection is performed between the convergence search result and the access search result to form the intersection result.

In one embodiment a user's organization is determined. An access control rule is retrieved based on the user's organization. In one embodiment the extensible markup language file is flattened.

FIG. 26 is a flow chart of the steps used in an control method in accordance with one embodiment of the invention. The process starts, step 680, by determining an access control rule for a user at step 682. In one embodiment the access control rule includes a query statement. A query is received from the user at step 684. An access control search is performed against the extensible markup language file to form an access control result at step 686. A query search is performed against the extensible markup language file to form a query search result at step 688. At step 690 the access control result is compared to the query search result to determine an allowed search result which ends the process at step 692. In one embodiment an access control query statement is created for the access control rule. In another embodiment, the query is converting into an executable stack. An executable stack is a series of simple commands. These commands can include simple search queries and operations on the results of the queries. In one embodiment, the access control query statement contains a predicate (608).

FIGs. 27 & 28 are a flow chart of the steps used in an access control method in accordance with one embodiment of the invention. The process starts, step 700, by receiving a query from a user at step 702. Next an access control rule associated with the user is determined at step 704. A query search on the extensible markup language file is performed at step 706. A query search result is stored at step 708. An access search on the extensible markup language file is performed at step 710. An access search result is stored at step 712. At step 714 the query search result and the

access search result are compared to determine an allowed search result which ends the process at step 716. In one embodiment a search convergence depth is compared to an access convergence depth. When the search convergence depth is equal to the access convergence depth, an intersection is performed between the query search result and the access search result to form an intersection result. When the access rule is a hide command, the allowed search result is a non-intersecting result. When the access result is a hide command, the allowed search result is a non-intersecting result. When the search convergence depth is greater than the access convergence depth, the query search result is converged to the access convergence depth to form a converged query search result. An intersection is performed between the convergence search result and the access search result to form the intersection result. When the search convergence depth is less than the access convergence depth, an intersection is performed between the query search result and the access search result to form the intersection result. Note that while the search results (access search) is commonly described as line numbers it could be data or metadata.

In one embodiment, the query is converted into an execution stack. Creating a line of the execution stack may include an operation, a convergence depth, a term and a pattern. The execution stack is explained in detail in FIG. 29. In one embodiment a query is converted into a plurality of lines of the execution stack. The lines are executed to form a plurality of results.

In one embodiment the extensible markup language file is flattened to form a flattened extensible markup language file. The query search on the flattened extensible markup language file returns a line number. An intersection is performed between a plurality of line numbers form the

query search result and a second plurality of numbers from the access search result.

FIG. 29 is an example of how a query is converted into an executable stack in accordance with one embodiment of the invention. A query 720 is shown that is searching for CDs with a price less than $9.75 and the artist is Bob Dylon. This query 720 is converted into an execution stack 722. The execution stack 722 has three lines each containing an operation (type) 724, a convergence depth 726, a term 728 and a pattern 730. The operation of the first line is "less than" (LT) 732. The convergence depth is three 734. The term is /ND/CAT/CD/PRICE> 736. The pattern is "9.75" 738. The first line will find all CDs with a price less than $9.75. The second line will find all CDs by the artist "Bob Dylon". The final line performs an AND operation between these two searches. The execution stack simplifies the process of performing complex searches.

Thus there has been described an access control method that is a flexible and easy to use as a query language. This provides an access control method that is considerably more flexible than the prior art, which are commonly restricted to access controls on directories. In addition, the access control rules can be written by anyone who understands the query language. Because of the ease and flexibility of the access control language, extensible markup language files or databases that are converted to extensible markup language files can be conglomerated together. This means that customer services files can be combined with accounting files and sales files for instance. The access control rules can assure that people who only need access to accounting information do not see sales information. This reduces the number of separate extensible markup language files that have to maintained.

FIG. 30 is a flow chart of the steps used in a control method in accordance with one embodiment of the invention. The process starts, step 750, by predefining a pattern as a trigger at step 752. Next a search for the pattern is initiated at step 754. When the pattern is found at step 756, altering the extensible markup language file which ends the process at step 758. In one embodiment, when the pattern is found, an extensible markup language document is created. In one embodiment, a command is a trigger. For instance a query command can act as the trigger or a delete command can act as a trigger. In another embodiment, the trigger may be a tag (metatag) or a data (data string). For instance, the tag "president" might prompt the action to compare the data with "Clinton". Another example might be that the data "Clinton" might prompt the action to search for the data "William". In one embodiment, the step of altering the extensible markup language file might be performing an access control search. Note that this only alters the user's access to the underlying file not the file itself however to the user this effectively alters the file. In another embodiment, the step of altering the extensible markup language file includes deleting a portion of the extensible markup language file. For instance, when a record of the file is deleted there may not be any records left related to the customer header information, thus the customer header information is deleted. A delete may also invoke a question of whether the person has authority to delete the information. In another embodiment, altering the extensible markup language file includes performing a translation of a portion of the extensible markup language file. For instance, a user may be looking at a file that has the tag "client" and the user's program may only recognize the tag "customer". The system would translate the tag "client" to the tag "customer".

Thus there has been described a control method that provides significant flexibility to extensible markup language files.

The methods described herein can be implemented as computer-readable instructions stored on a computer-readable storage medium that when executed by a computer will perform the methods described herein.

While the invention has been described in conjunction with specific embodiments thereof, it is evident that many alterations, modifications, and variations will be apparent to those skilled in the art in light of the foregoing description. Accordingly, it is intended to embrace all such alterations, modifications, and variations in the appended claims.